Deep Learning
MSc in Computer Science and Engineering
MSc in Electrical and Computer Engineering

Final exam — February 10, 2023

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 14 pages and is not missing any sheets, then write your full name and student n. on this page (and your number in all others).
- The test has a total of 17 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer. Please use a pen, not a pencil.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	Part 2	Part 3, Pr. 1	Part 3, Pr. 2	Total
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

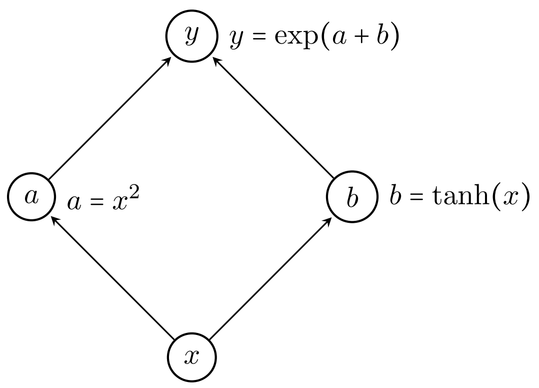
In each of the following questions, indicate your answer by *checking a single option.*

1. (4 points) Which framework is most suitable for a classification task with 3 labels?
   - ☐ Linear regression.
   - ☐ Binary logistic regression.
   - ■ **Multi-class logistic regression.**
   - ☐ None of the above is suitable.

2. (4 points) Which activation function best handles the problem of vanishing gradients?
   - ☐ Softmax.
   - ☐ Hyperbolic tangent.
   - ☐ Sigmoid function.
   - ■ **Rectified linear unit.**

3. (4 points) Which of the following statements about autoregressive RNN-based language models is correct?
   - ☐ Just like n-gram models, RNNs make a Markov assumption, hence they can only remember the last n words they generate, for a given $n \in \mathbb{N}$.
   - ☐ RNNs have unbounded memory, however they have a tendency to "remember" less accurately the most recent words they generate.
   - ■ **RNNs have unbounded memory, however for long sequences they have a tendency to "remember" less accurately the initial words they have generated.**
   - ☐ RNNs suffer from vanishing gradients, but this can be mitigated with gradient clipping.

4. (4 points) Consider the following computation graph. What is the derivative of y with respect to x?



   - ■ $\boldsymbol{\left(2x + 1 - \tanh^2(x)\right) y}$.
   - ☐ $(2x + \tanh(x)\,(1 - \tanh(x)))\exp(a + b)$.
   - ☐ $2x + 1 - \tanh^2(x)$.
   - ☐ $\exp(i\pi) + 1$.

Solution: It is $\big(2x + 1 - \tanh^2(x)\big)\, y$:

$$\begin{aligned}
&\frac{\partial y}{\partial a} = \frac{\partial y}{\partial b} = \exp(a+b) = y, \\
&\frac{\partial a}{\partial x} = 2x, \quad \frac{\partial b}{\partial x} = 1 - \tanh^2(x), \\
&\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} = \big(2x + 1 - \tanh^2(x)\big)\, y.
\end{aligned}$$

5. (4 points) Let $\boldsymbol{s} \in \mathbb{R}^{10000}$ be a vector of logits where $s_1 = 0$ and $s_j = 1$, for all $j \in \{2, 3, \ldots, 10000\}$. A probability vector $\boldsymbol{p} = [p_1, \ldots, p_{10000}]^\top$ is obtained by taking the softmax transformation of $\boldsymbol{s}$. What is the value of p_1?

   - ☐ 0.
   - ☐ 1.
   - ☐ 10000^{-1}.
   - ■ **None of the above.**

   Solution: We have $p_1 = 1/(1 + 9999 \exp(1))$, which does not match any of the first three options.

6. (4 points) Assume we train an LSTM model on English to Portuguese translation. We ensure the training dataset contains the same amount of masculine/feminine pronouns. The resulting model:

   - ☐ Is not biased since the training dataset was balanced with respect to gender pronoun frequencies.
   - ☐ Can still be biased if the model was not trained for enough epochs.
   - ■ **Can still be biased if the co-occurence frequency of masculine/feminine pronouns with non-gendered words such as {doctor, nurse} is not balanced in the training data.**
   - ☐ Can still biased because LSTM cells include a bias vector.

   Solution: The marginal frequency of male/female pronouns is not sufficient to ensure that there is no bias. It is important to check the co-occurrence with non-gendered words.

7. (4 points) Consider a sequence of length L leading to a matrix of token representations $\boldsymbol{X} \in \mathbb{R}^{L \times D}$. What is the output of a self-attention layer when $\boldsymbol{X}$ is provided as input, using only a single attention head and where the projection matrices are $\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = \boldsymbol{I}$ (the identity matrix)?

   - ☐ A vector representation in $\mathbb{R}^D$ that summarizes the most relevant vectors of $\boldsymbol{X}$ through an attention layer in which the query is the first row of $\boldsymbol{X}$.
   - ☐ A vector representation in $\mathbb{R}^L$ that summarizes the most relevant vectors of $\boldsymbol{X}$ through an attention layer in which the query is the first column of $\boldsymbol{X}$.
   - ☐ The sequence is fed through an LSTM and the last hidden state $\boldsymbol{h}_L \in \mathbb{R}^D$ is considered the output of the self-attention layer.
   - ■ **A matrix representation in $\mathbb{R}^{L \times D}$, where each row $\boldsymbol{z}_i \in \mathbb{R}^D$ results from doing attention over the whole original sequence in which the query is $\boldsymbol{x}_i \in \mathbb{R}^D$.**

8. (4 points) Which of the following statements is true?

   - ☐ The family of encoder-decoder large language models, of which T5 is an example, is suitable for classification tasks but unsuitable for generation tasks.
   - ■ **Encoder-only models can be pretrained with masked language modeling. An example is BERT.**
   - ☐ Decoder-only models are usually pretrained with masked language modeling. An example is GPT.

- ☐ Prompting is a more efficient way to adapt models than fine-tuning, but it can only be done if the pretrained model is trained with a masked language modeling objective.

Solution: The family of encoder-decoder large language models is suitable to both classification and generation. Decoder-only models, including GPT, are trained with a causal language modeling objective. Prompting is a more efficient way to adapt models than fine-tuning, but it is usually done with models such as GPT, which are trained with a causal language modeling objective.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) Explain how a bidirectional RNN works and when they are useful.

   Solution: Bidirectional RNNs combine a left-to-right RNN with a right-to-left RNN. After propagating the states in both directions, the states of the two RNNs are concatenated. Bidirectional RNNs are used as sequence encoders, their main advantage is that each state contains contextual information coming from both sides. Unlike unidirectional RNNs, they have the same focus on the beginning and on the end of the input sequence.

2. (6 points) Explain the difference between self-attention and masked self-attention, as well as why and where the masked version is used.

   Solution: The self-attention in transformers allows any word to attend to any other word, both in the source and on the target. When the model is generating a sequence left-to-right it cannot attend at future words, which have not been generated yet. At training time, causal masking is needed in the decoder self-attention to mask future words, to reproduce test time conditions.

3. (6 points) Explain what an adapter is and what the advantages over fine-tuning are.

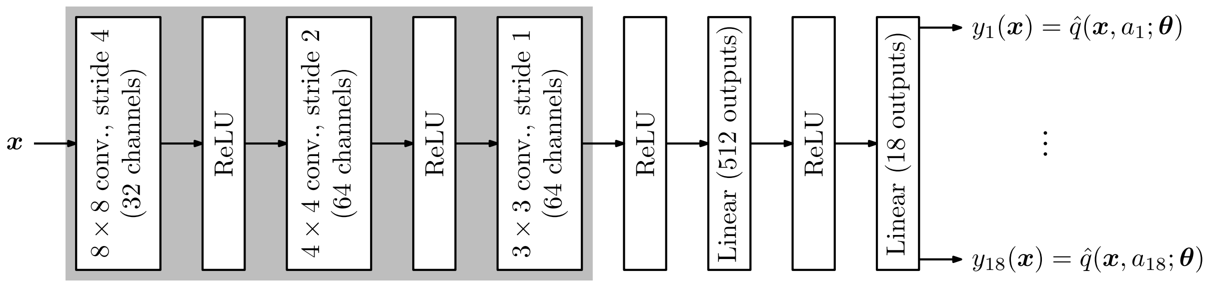
   Solution: TO DO

Part 3: Problems (50 points)

Problem 1: DQN (25 points)

The *Deep Q network* (DQN) is a convolutional neural network proposed in a pioneer work by DeepMind that first combined deep convolutional networks with reinforcement learning. DQN was used to learn how to play a number of games from the Atari 2600 console using only information from the pixels and the points in the game. DQN is summarized in Fig. 1, where the nonlinearities have been explicitly represented.

The input to the network consists of an image $\boldsymbol{x}$ with 84×84 pixels and 4 channels. The network has 18 outputs, where output i corresponds to a scalar value $\hat{q}(\boldsymbol{x}, a_i; \boldsymbol{\theta})$; $\boldsymbol{\theta}$ represents the parameters in the network and a_i is one among 18 possible actions in the game. Let $\mathcal{A}$ denote the set of such actions.



$\boldsymbol{x}$
8×8 conv., stride 4 (32 channels)
ReLU
4×4 conv., stride 2 (64 channels)
ReLU
3×3 conv., stride 1 (64 channels)
ReLU
Linear (512 outputs)
ReLU
Linear (18 outputs)
$y_1(\boldsymbol{x}) = \hat{q}(\boldsymbol{x}, a_1; \boldsymbol{\theta})$
$y_{18}(\boldsymbol{x}) = \hat{q}(\boldsymbol{x}, a_{18}; \boldsymbol{\theta})$

Figure 1: DQN architecture.

In very broad terms, to train the network, a set of samples of the form $(\boldsymbol{x}, a, target)$ is collected, where *target* is a scalar value and $a \in \mathcal{A}$. The loss associated with one such sample is then given by

$$L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) = \frac{1}{2}(target - \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))^2. \tag{1}$$

1. (10 points) Compute the total number of parameters in the network. Do not forget to account for the bias terms. Indicate all relevant computations.

   Solution: We have:

   - In the first convolutional layer, the number of parameters is $(8 \times 8 \times 4 + 1) \times 32 = 8,224$. The output has a dimension of $20 \times 20 \times 32$.
   - In the second convolutional layer, the number of parameters is $(4 \times 4 \times 32 + 1) \times 64 = 32,832$. The output has a dimension of $9 \times 9 \times 64$.
   - In the third convolutional layer, the number of parameters is $(3 \times 3 \times 64 + 1) \times 64 = 36,928$. The output has a dimension of $7 \times 7 \times 64$.

   The (flattened) input to the first linear layer is a vector with $7 \times 7 \times 64 = 3,136$ elements. The first linear layer has, therefore, a total of $3,136 \times 512 + 512 = 1,606,144$ parameters. Finally, the output layer has a total of $512 \times 18 + 18 = 9,234$ parameters.

   Summarizing, the network has a total of $8,224 + 32,832 + 36,928 + 1,606,144 + 9,234 = 1,693,362$ parameters.

2. (7 points) Suppose that you want to replace the layers in the shaded block in Fig. 1 by a single linear layer. Compute the number of parameters of the resulting neural network and compare them with the number of parameters of the original network.

   Note: If you did not answer Question 1, consider that the number of parameters of the original network is, approximately, 1.7×10^6 parameters.

   Solution: As seen in 1, the dimension of the output of the shaded is $7 \times 7 \times 64$ or, when flattened, $3,136 \times 1$. If we replace the shaded block with a linear layer, the input for such layer would have a dimension $(84 \times 84 \times 4) \times 1 = 28,224 \times 1$. The total number of parameters in that linear layer would thus be $28,224 \times 3,136 = 88,510,464$.

   The total number of parameters of the resulting network would thus be $88,510,464 + 1,606,144 + 9,234 = 90,125,842$ parameters, which is over 53 times larger than the original network.

3. (8 points) Suppose that, given a sample $(\boldsymbol{x}, a, target)$, we let

$$g_k = \frac{\partial L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k},$$

where $L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))$ is the loss defined in (1). Further suppose you want to include L_2 regularization when training the network. Indicate how the loss $L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))$ should be modified to include such regularization, and compute the derivative with respect to θ_k as a function of g_k.

Solution: The new loss would be

$$L_{\text{new}}(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) = L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta})) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2,$$

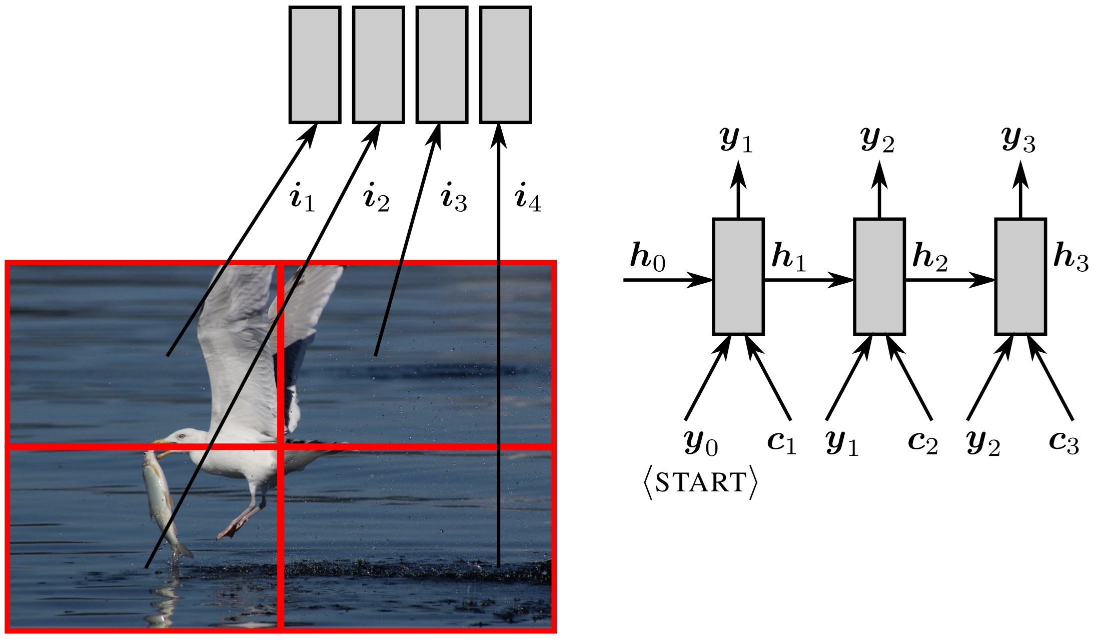
where N is the number of points in the dataset and $\|\cdot\|_2$ indicates the 2-norm of a vector, defined for an arbitrary vector $\boldsymbol{u}$ as $\|\boldsymbol{u}\|_2^2 = \boldsymbol{u}^T\boldsymbol{u}$.

The derivative with respect to θ_k becomes

$$\begin{aligned}\frac{\partial L_{\text{new}}(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k} &= \frac{\partial L(target, \hat{q}(\boldsymbol{x}, a; \boldsymbol{\theta}))}{\partial \theta_k} + \lambda\theta_k \\ &= g_k + \lambda\theta_k.\end{aligned}$$

Problem 2: Image Captioning (25 points)

Consider the problem shown in the figure, where an image is given and the task is to generate a descriptive natural language caption for the image.



The image is processed by a convolutional neural network (not represented), resulting in 4 feature representations $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, $\boldsymbol{i}_3$, $\boldsymbol{i}_4$, where each $\boldsymbol{i}_j \in \mathbb{R}^2$ as shown in the figure. Then, the caption is generated auto-regressively by an RNN-based decoder, conditioned on the image feature representations. The output vocabulary contains only 6 words, including the ⟨START⟩ and ⟨STOP⟩ symbols, with the following embedding vectors:

$$\boldsymbol{y}_{\langle\text{START}\rangle} = [0,0,0]^\top, \qquad \boldsymbol{y}_{\langle\text{STOP}\rangle} = [1,1,1]^\top, \qquad \boldsymbol{y}_{\text{fish}} = [-1,2,0]^\top,$$
$$\boldsymbol{y}_{\text{seagull}} = [1,-2,0]^\top, \qquad \boldsymbol{y}_{\text{flying}} = [0,-1,-1]^\top, \qquad \boldsymbol{y}_{\text{fishing}} = [0,2,1]^\top.$$

1. (10 points) Let the input image be the one shown in the figure, with the following image feature maps:

$$\boldsymbol{i}_1 = [4,0]^\top, \qquad \boldsymbol{i}_2 = [0,4]^\top, \qquad \boldsymbol{i}_3 = [0,0]^\top, \qquad \boldsymbol{i}_4 = [0,0]^\top.$$

At each time step t, the RNN-based decoder receives as input $\boldsymbol{x}_t \in \mathbb{R}^5$, which is a **concatenation** of the previous output embedding $\boldsymbol{y}_{t-1} \in \mathbb{R}^3$ and an image representation $\boldsymbol{c}_t \in \mathbb{R}^2$ (by this order), and uses this input and the previous hidden state $\boldsymbol{h}_{t-1}$ to compute the new state $\boldsymbol{h}_t$. This is followed by a linear output layer with hidden-to-output matrix

$$\boldsymbol{W}_{yh} = \left[\begin{array}{rr} -5 & -5 \\ 0 & 3 \\ 1 & 2 \\ 2 & 2 \\ 3 & -1 \\ 2.9 & 0 \end{array}\right] \begin{array}{l} \#\ \langle\text{START}\rangle \\ \#\ \langle\text{STOP}\rangle \\ \#\ \text{fish} \\ \#\ \text{seagull} \\ \#\ \text{flying} \\ \#\ \text{fishing} \end{array},$$

where each row of this matrix corresponds to the words stated above. The input-to-hidden matrix and the recurrence matrix are given respectively by

$$\boldsymbol{W}_{hx} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{W}_{hh} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Assume that the RNN uses `relu` activations, that all biases are vectors of zeros, and that the initial hidden state $\boldsymbol{h}_0$ is a vector of zeros.

In this question, assume that $\boldsymbol{c}_t := \boldsymbol{c}$ is **constant for all time steps** and obtained through average pooling of the image feature representations, $\boldsymbol{c} = \frac{1}{4}\sum_{j=1}^{4} \boldsymbol{i}_j$, without any attention mechanism. Compute the first three words of the caption using **greedy decoding**.

Solution: We have:

$$
\begin{aligned}
\boldsymbol{c} &= \frac{1}{4}\sum_{j=1}^{4}\boldsymbol{i}_j = \frac{1}{4}[4,4]^\top = [1,1]^\top.\\
\boldsymbol{x}_1 &= \text{concat}(\boldsymbol{y}_{\langle\text{START}\rangle}, \boldsymbol{c}) = [0,0,0,1,1]^\top.\\
\boldsymbol{h}_1 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_1 + \boldsymbol{W}_{hh}\boldsymbol{h}_0)\\
&= \text{relu}([1,1]^\top + [0,0]^\top) = [1,1]^\top.\\
y_1 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_1)\\
&= \text{argmax}([-10,3,3,4,2,2.9]) = 4 \quad \Rightarrow \quad \text{seagull.}\\
\boldsymbol{x}_2 &= \text{concat}(\boldsymbol{y}_{\text{seagull}}, \boldsymbol{c}) = [1,-2,0,1,1]^\top.\\
\boldsymbol{h}_2 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_2 + \boldsymbol{W}_{hh}\boldsymbol{h}_1)\\
&= \text{relu}([2,-1]^\top + [1,1]^\top) = [3,0]^\top.\\
y_2 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_2)\\
&= \text{argmax}([-15,0,3,6,9,8.7]) = 5 \quad \Rightarrow \quad \text{flying.}\\
\boldsymbol{x}_3 &= \text{concat}(\boldsymbol{y}_{\text{flying}}, \boldsymbol{c}) = [0,-1,-1,1,1]^\top.\\
\boldsymbol{h}_3 &= \text{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_3 + \boldsymbol{W}_{hh}\boldsymbol{h}_2)\\
&= \text{relu}([0,-1]^\top + [0,3]^\top) = [0,2]^\top.\\
y_3 &= \text{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_3)\\
&= \text{argmax}([-10,6,4,4,-2,0]) = 2 \quad \Rightarrow \quad \langle\text{STOP}\rangle.
\end{aligned}
$$

2. (5 points) Assume now that, instead of using a constant $\boldsymbol{c}_t$ for all time steps, the RNN-based decoder has a **scaled dot-product** attention mechanism that attends to the image feature representations. For each time step, the query vector is $\boldsymbol{h}_{t-1}$ and the image feature representations $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, $\boldsymbol{i}_3$, $\boldsymbol{i}_4$ are used as both keys and values. Assume again that $\boldsymbol{h}_0$ is a vector of zeros. For the first time step ($t = 1$), compute the attention probabilities and the resulting image vector $\boldsymbol{c}_1$. Will the first word be the same or different from the one in the previous question?

Solution: We have:

$$
\begin{aligned}
\boldsymbol{I} &= [\boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_3, \boldsymbol{i}_4]^\top = \begin{bmatrix} 4 & 0 \\ 0 & 4 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.\\
\boldsymbol{p}_1 &= \text{softmax}\left(\frac{1}{\sqrt{2}}\boldsymbol{I}\boldsymbol{h}_0\right) = [.25,.25,.25,.25]^\top.\\
\boldsymbol{c}_1 &= [1,1]^\top.
\end{aligned}
$$

The first word will be the same ("seagull") since the attention mechanism gives uniform probabilities due to the query being all-zeros, hence the first step is exactly the same as before.

3. (10 points) Compute the second word of the caption using the attention-based RNN decoder.

Solution: The first word is "seagull", as seen before, and the first state is $\boldsymbol{h}_1 = [1, 1]$ as in the first exercise. For the second and time step, we have:

$$
\begin{aligned}
\boldsymbol{p}_2 &= \operatorname{softmax}(\boldsymbol{I}\boldsymbol{h}_1) = \operatorname{softmax}(\frac{1}{\sqrt{2}}[4,4,0,0]^\top) = [0.47209639, 0.47209639, 0.02790361, 0.02790361]^\top. \\
\boldsymbol{c}_2 &= \boldsymbol{I}^\top \boldsymbol{p}_2 = [1.88838556, 1.88838556]^\top. \\
\boldsymbol{x}_2 &= \operatorname{concat}(\boldsymbol{y}_{\text{seagull}}, \boldsymbol{c}) = [1, -2, 0, 1.888, 1.888]^\top. \\
\boldsymbol{h}_2 &= \operatorname{relu}(\boldsymbol{W}_{hx}\boldsymbol{x}_2 + \boldsymbol{W}_{hh}\boldsymbol{h}_1) \\
&= \operatorname{relu}([2.888, -0.111]^\top + [1, 1]^\top) = [3.888, 0.888]^\top. \\
y_2 &= \operatorname{argmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_2) \\
&= \operatorname{argmax}([-23.9, 2.7, 5.7, 9.6, 10.8, 11.3]) = 6 \quad \Rightarrow \quad \text{fishing}.
\end{aligned}
$$